Offline Constrained Backward Time Travel Planning

Quentin Bramas, Jean-Romain Luttringer, Sébastien Tixeuil

Offline Constrained Backward Time Travel Planning

Quentin Bramas[1], Jean-Romain Luttringer[1], and Sébastien Tixeuil[2,3]

[1] ICUBE, Strasbourg University, CNRS, Strasbourg, France
[2] Sorbonne University, CNRS, LIP6, Paris, France
[3] Institut Universitaire de France, Paris, France

This version of the article has been accepted for publication in the proceedings of the International Symposium on Stabilizing, Safety, and Security of Distributed Systems (SSS 2023), after peer review and is subject to Springer Nature's AM terms of use, but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at:
https://doi.org/10.1007/978-3-031-44274-2_35

Abstract. We model transportation networks as dynamic graphs and introduce the ability for agents to use Backward Time-Travel (BTT) devices at any node to travel back in time, subject to certain constraints and fees, before resuming their journey.
We propose exact algorithms to compute travel plans with constraints on BTT cost or the maximum time that can be traveled back while minimizing travel delay (the difference between arrival and starting times). These algorithms run in polynomial time. We also study the impact of BTT device pricing policies on the computation of travel plans with respect to delay and cost and identify necessary properties for pricing policies to enable such computation.

1 Introduction

Evolving graphs (and their many variants) are graphs that change over time and are used to model real-world systems that evolve. They have applications in many fields in Computer Science, where they arise in areas such as compilers, databases, fault-tolerance, artificial intelligence, and computer networks. To date, such graphs were studied under the hypothesis that time can be traveled in a single direction (to the future, by an action called waiting), leading to numerous algorithms that revisit static graph notions and results.

In this paper, we introduce the possibility of Backward time travel (BTT) (that is, the ability to go back in time) when designing algorithms for dynamic graphs. In more details, we consider the application of BTT devices to transportation networks modeled by evolving graphs. In particular we focus on the ability to travel from point A to point B with minimal delay (that is, minimizing the time difference between arrival and start instants), taking into account

meaningful constraints, such as the cost induced by BTT devices, or their span (how far back in time you are allowed to go).

To this paper, BTT was mostly envisioned in simple settings (with respect to the cost associated to time travel or its span). For example, the AE model [12] considers that a single cost unit permits to travel arbitrarily in both space and time, trivializing the space-time travel problem entirely. Slightly more constrained models such as TM [11] and BTTF [16] consider devices that either: *(i)* only permit time travel [11] (but remain at the same position), or *(ii)* permit either time travel or space travel, but not both at the same time [16]. However, the cost involved is either null [11], or a single cost unit per time travel [16].

Instead, we propose to discuss BTT in a cost-aware, span-aware context, that implies efficiently using BTT devices within a transportation system (from a simultaneous delay and cost point of view), and the computation of the corresponding multi-modal paths. More precisely, in this paper, we address the problem of space-time travel planning, taking into account both the travel delay of the itinerary and the cost policy of BTT device providers. The context we consider is that of transportation systems, where BTT devices are always available to the agents traveling. Using each BTT device has nevertheless a cost, decided by the BTT device provider, and may depend on the span of the backward time jump. Although BTT devices are always active, the ability to go from one location to another (that is, from one BTT device to another) varies across time. We consider that this ability is conveniently modeled by a dynamic graph, whose nodes represent BTT devices, and whose edges represent the possibility to instantly go from one BTT device to another. Given a dynamic graph, we aim at computing travel plans, from one BTT device to another (the closest to the agent's actual destination), considering not only travel delay and induced cost, but also schedule availability and common limitations of BTT devices.

In the following, we study the feasibility of finding such travel plans, depending on the pricing policy. It turns out that when the schedule of connections is available (that is, the dynamic graph is known), very loose conditions on the pricing policy enable to devise optimal algorithms (with respect to the travel delay and induced cost) in polynomial time, given a cost constraint for the agents, or a span constraint for the BTT devices.

Related Work. Space-Time routing has been studied, but assuming only forward time travel, *i.e.*, waiting, is available. The idea of using dynamic graphs to model transportation network was used by many studies (see *e.g.* Casteigts et al.[2] and references herein), leading to recently revisit popular problems previously studied in static graphs [1,4,10]. In a dynamic (or temporal) graph, a journey represents a temporal path consisting in a sequence of edges used at given non-decreasing time instants. The solvability of a problem can depend on whether or not a journey can contain consecutive edges occurring at the same time instant. Such journeys are called *non-strict*, as opposed to *strict* journey where the sequence of time instants must be strictly increasing. In our work, we extend the notion of non-strict journey to take into account the possibility to go back in time at each node, but one can observe that our algorithm also work

with the same extension for strict journey by adding one time unit to the arrival of each edge in our algorithms.

The closest work in this research path is due to Casteigts et al [3], who study the possibility of discovering a temporal path between two nodes in a dynamic network with a waiting time constraint: at each step, the traveling agent cannot wait more than c time instants, where c is a given constant. It turns out that finding the earliest arriving such temporal path can be done in polynomial time. Perhaps surprisingly, Villacis-Llobet et al [14] showed that if one allows to go several times through the same node, the obtained temporal path can arrive earlier, and finding it can be done in linear time. As previously mentioned, this line of work only considers *forward* time travel: a temporal path cannot go back in time.

Constrained-shortest-paths computation problems have been extensively studied in the context of static graphs [5]. Although these problems tend to be NP-Hard [7] (even when considering a single metric), the ones considering two additive metrics (commonly, the delay and a cost) gained a lot of traction over the years due to their practical relevance, the most common use-case being computer networks [8,9]. In this context, each edge is characterized by a weight vector, comprising both cost and delay. Path computation algorithms thus have to maintain and explore all non-comparable paths, whose number may grow exponentially with respect to the size of the network. To avoid a worst-case exponential complexity, most practical algorithms rely on either approximation schemes [13] or heuristics. However, these contributions do not study multi-criteria path computation problems within a time travel context. Conversely, we study and provide results regarding the most relevant time-traveling problems while considering the peculiarities of this context (in particular, the properties of the cost function). In addition, we show that most of these problems can be solved optimally in polynomial-time.

Contributions. In this paper, we provide the following contributions:

- An in-depth analysis of the impact of the BTT device providers pricing policies on the computation of low-latency and low-cost paths. In particular, we show that few features are required to ensure that the efficient computation of such paths remains possible.
- Two exact polynomial algorithms able to compute travels with smallest delay to a given destination and minimizing the cost of traveling back in time. The first algorithm also supports the addition of a constraint on the backward cost of the solution. The other one supports a constraint on how far back in the past one can go at each given time instant.

2 Model

In this section, we define the models and notations used throughout this paper, before formalizing the aforementioned problems.

We represent the network as an evolving graph, as introduced by Ferreira [6]: a graph-centric view of the network that maps a dynamic graph as a sequence of static graphs. The *footprint* of the dynamic graph (that includes all nodes and edges that appear at least once during the lifetime of the dynamic graph), is fixed. Furthermore, we assume that the set of nodes is fixed over time, while the set of edges evolves.

More precisely, an evolving graph G is a pair $(V, (E_t)_{t\in\mathbb{N}})$, where V denotes the finite set of vertices, $\mathbb{N}$ is the infinite set of time instants, and for each $t \in \mathbb{N}$, E_t denotes the set of edges that appears at time t. The *snapshot* of G at time t is the static graph $G(t) = (V, E_t)$, which corresponds to the state, supposedly fixed, of the network in the time interval $t, t+1)$. The *footprint* $\mathcal{F}(G)$ of G is the static graph corresponding to the union of all its snapshots, $\mathcal{F}(G) = \left(V, \bigcup_{t\in\mathbb{N}} E_t\right)$. We say $((u, v), t)$ is a temporal edge of graph G if $(u, v) \in E_t$. We say that an evolving graph is *connected* if its footprint is connected.

Space-time Travel. We assume that at each time instant, an agent can travel along any number of adjacent consecutive communication links. However, the graph may not be connected at each time instant, hence it may be that the only way to reach a particular destination node is to travel forward (*i.e.*, wait) or backward in time, to reach a time instant where an adjacent communication link exists. In more detail, an agent travels from a node s to a node d using a *space-time travel* (or simply travel when it is clear from the context).

Definition 1. *A* space-time travel *of* length k *is a sequence* $((u_0, t_0), (u_1, t_1), \ldots, (u_k, t_k))$ *such that*

- $\forall i \in \{0, \ldots k\}$, $u_i \in V$ *is a node and* $t_i \in \mathbb{N}$ *is a time instant,*
- $\forall i \in \{0, \ldots k-1\}$, *if* $u_i \neq u_{i+1}$, *then* $t_i = t_{i+1}$ *and* $(u_i, u_{i+1}) \in E_{t_i}$ i.e.*, there is a temporal edge between* u_i *and* u_{i+1} *at time* t_i.

By extension, the *footprint* of a travel is the static graph containing all edges (and their adjacent nodes) appearing in the travel. Now, the *itinerary* of a travel $((u_0, t_0), (u_1, t_1), \ldots, (u_k, t_k))$ is its projection $(u_0, u_1, \ldots, u_k)$ on nodes, while its *schedule* is its projection $(t_0, t_1, \ldots, t_k)$ on time instants.

Definition 2. *A travel* $((u_0, t_0), (u_1, t_1), \ldots, (u_k, t_k))$ *is* simple *if for all* $i \in \{2, \ldots, k\}$ *and* $j \in \{0, \ldots, i-2\}$, *we have* $u_i \neq u_j$.

Intuitively, a travel is simple if its footprint is a line (*i.e.*, a simple path) and contains at most one time travel per node (as a consequence, no node appears three times consecutively in a simple travel).

Definition 3. *The* delay *of a travel* $T = ((u_0, t_0), (u_1, t_1), \ldots, (u_k, t_k))$, *denoted* $delay(T)$ *is defined as* $t_k - t_0$.

The Backward cost of a travel.

Definition 4. *The* backward-cost *is the cost of going to the past. The backward-cost function* $\mathfrak{f} : \mathbb{N}^* \to \mathbb{R}^+$ *returns, for each* $\delta \in \mathbb{N}$*, the backward-cost* $\mathfrak{f}(\delta)$ *of traveling* δ *time instants to the past. As we assume that there is no cost associated to forward time travel (that is, waiting), we extend* $\mathfrak{f}$ *to* $\mathbb{Z}$ *by setting* $\mathfrak{f}(-\delta) = 0$*, for all* $\delta \in \mathbb{N}$*. In particular, the backward-cost of traveling* 0 *time instants in the past is zero. When it is clear from context, the backward-cost function is simply called the cost function.*

Definition 5. *The* backward-cost *(or simply cost) of a travel* $T = ((u_0, t_0), (u_1, t_1), \ldots, (u_k, t_k))$*, denoted* $cost(T)$ *is defined as follows:*

$$cost(T) = \sum_{i=0}^{k-1} \mathfrak{f}(t_i - t_{i+1})$$

Definition 6. *Let* $T_1 = ((u_0, t_0), (u_1, t_1), \ldots, (u_k, t_k))$ *and* $T_2 = ((u'_0, t'_0), (u'_1, t'_1), \ldots, (u'_{k'}, t'_{k'}))$ *be two travels. If* $(u_k, t_k) = (u'_0, t'_0)$*, then the* concatenated travel $T_1 \oplus T_2$ *is defined as follows:*

$$T_1 \oplus T_2 = ((u_0, t_0), (u_1, t_1), \ldots, (u_k, t_k), (u'_1, t'_1), \ldots, (u'_{k'}, t'_{k'}))$$

Remark 1. One can easily prove that $cost(T_1 \oplus T_2) = cost(T_1) + cost(T_2)$. In the following, we sometimes decompose a travel highlighting an intermediate node: $T = T_1 \oplus ((u_i, t_i)) \oplus T_2$. Following the definition, this means that T_1 ends with (u_i, t_i), and T_2 starts with (u_i, t_i), so we also have $T = T_1 \oplus T_2$ and $cost(T) = cost(T_1) + cost(T_2)$.

Our notion of space-time travel differs from the classical notion of *journey* found in literature related to dynamic graphs [6] as we do *not* assume time instants monotonically increase along a travel. As a consequence, some evolving graphs may not allow a journey from A to B yet allows one or several travels from A to B (See Figure 3).

We say a travel is cost-optimal if there does not exist a travel with the same departure and arrival node and times as T having a smaller cost. One can easily prove the following Property.

Property 1. Let T be a cost-optimal travel from node u to node v arriving at time t, and T' a sub-travel of T *i.e.*, a travel such that $T = T_1 \oplus T' \oplus T_2$. Then T' is also cost-optimal. However, this is not true for delay-optimal travels.

Problem specification. We now present the problems that we aim to solve in this paper. First, we want to arrive at the destination as early as possible, *i.e.*, finding a time travel that minimizes the delay. Among such travels, we want to find one that minimizes the backward cost.

In the remaining of this paper, we consider a given evolving graph $G = (V, (E_t)_{t \in \mathbb{N}})$, a given a cost function $\mathfrak{f}$, a source node *src* and a destination node *dst* in V. $Travels(G, src, dst)$ denotes the set of travels in G starting from *src* at time 0 and arriving at *dst*.

Definition 7. *The* Optimal Delay Optimal Cost space-time travel planning *(ODOC) problem consists in finding, among all travels in* $Travels(G, src, dst)$*, the ones that minimize the travel delay and, among them, minimize the cost. A solution to the ODOC problem is called an ODOC travel.*

One can notice that this problem is not very hard as there is a single metric (the cost) to optimize, because a travel with delay zero always exists (if the graph is temporally connected). But in this paper we study the two variants defined thereafter (see the difference in bold).

Definition 8. *The* $\mathcal{C}$***-cost-constrained*** *ODOC problem consists in finding among all travels in* $Travels(G, src, dst)$ ***with cost at most*** $\mathcal{C} \geq 0$*, the ones that minimize the travel delay and, among them, one that minimizes the cost.*

Definition 9. *The* $\mathcal{H}$***-history-constrained*** *ODOC problem consists in finding among all travels in* $Travels(G, src, dst)$ ***satisfying,***

$$\forall u, u', t, t', \text{ if } T = T_1 \oplus ((u,t)) \oplus T_2 \oplus ((u',t')) \oplus T_3, \text{ then } t' \geq t - \mathcal{H},$$

the ones that minimize the travel delay and, among them, one that minimizes the cost.

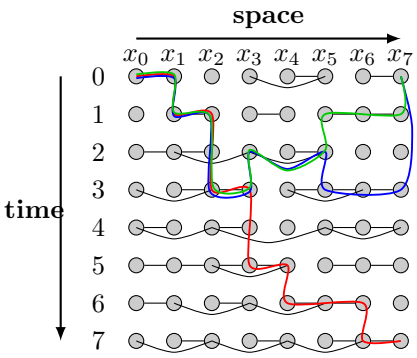

Fig. 1: Possible representation of an evolving graph. Possible travels from x_0 to x_7 are shown in red, green and blue. Note that the blue and green travels require to send an agent to the past (to a previous time instant).

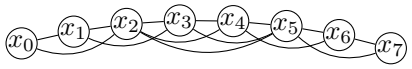

Fig. 2: Footprint of the evolving graph represented in Figure 1.

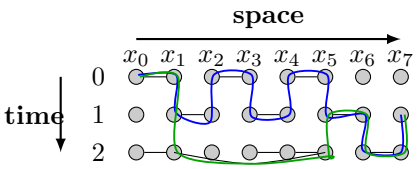

Fig. 3: Example of an evolving graph for which there exists no journey, yet there exists several travels from x_0 to x_7. The two travels, in blue and green, are 1-history-constrained.

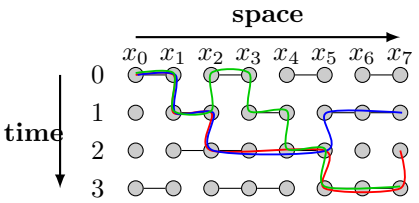

Fig. 4: Example of an evolving graph for which there exist at least three travels from x_0 to x_7 with a cost constraint of 1 (assuming $\mathfrak{f} : d \mapsto d$). The blue travel has optimal delay.

Visual representation of space-time travels.. To help visualize the problem, consider a set of $n + 1$ nodes denoted $x_0, x_1, x_2, \ldots, x_n$. Then, the associated evolving graph can be seen as a vertical sequence of graphs mentioning for each time instant which edges are present. A possible visual representation of an evolving graph can be seen in Figure 1. One can see the evolution of the topology (consisting of the nodes x_0 to x_7) over time through eight snapshots performed from time instants 0 to 7. Several possible travels are shown in red, green and blue. The red travel only makes use of forward time travel (that is, waiting) and is the earliest arriving travel in this class (arriving at time 7). The green and blue travels both make use of backward time travel and arrive at time 0, so they have minimal travel delay. Similarly, the red travel concatenated with $((x_7, 7), (x_7, 0))$ (*i.e.*, a backward travel to reach x_7 at time 0) also has minimal travel delay. However, if we assume that the cost function is the identity ($\mathfrak{f} : d \mapsto d$) then the green travel has a backward cost of 3, the blue travel has a backward cost of 4, and the concatenated red travel has a backward cost of 7. Adding constraints yields more challenging issues: assuming $\mathfrak{f} : d \mapsto d$ and a maximal cost $\mathcal{C}$ of 1, at least three travels can be envision for the evolving graph depicted in Figure 4, but finding the 1-cost-constrained travel that minimizes the delay (that is, the blue travel) is not as straightforward in this case, even if the footprint of the evolving graph is a line.

Similarly, in Figure 3 we show two $\mathcal{H}$-history-constrained travels, with $\mathcal{H} = 1$ (assuming $\mathfrak{f} : d \mapsto d$). Here, clearly, the green travel is optimal with a cost of 2 (the blue travel has cost 3). The choice made by the green travel to wait at node x_1 two time instants is good, even if it prevents future backward travel to time 0 since $\mathcal{H} = 1$; because it is impossible to terminates at time 0 anyway. So it seems like the choice made at node x_1 is difficult to make before knowing what is the best possible travel. If we add more nodes to the graph and repeat this kind of choice, we can create a graph with an exponential number of 1-history-constraint travel and finding one that minimizes the cost is challenging. Surprisingly, we show that it remains polynomial in the number of nodes and edges.

3 Backward-cost Function Classes

The cost function $\mathfrak{f}$ represents the cost of going back to the past. Intuitively, it seems reasonable that the function is non-decreasing (travelers are charged more it they go further back in time), however we demonstrate that such an assumption is not necessary to enable travelers to derive optimal cost space-time travel plans. As a matter of fact, the two necessary conditions we identify to optimally solve the ODOC space-time travel planning problem are $\mathfrak{f}$ to be non-negative and that it attains its minimum (not just converge to it). These conditions are shown to be sufficient by construction, thanks to the algorithm presented in the next section (and Theorem 2). Due to space constrains, proofs are omitted.

Definition 10. *A cost function $\mathfrak{f}$ is* user optimizable *if it is non-negative, and it attains its minimum when restricted to any interval $[C, \infty)$, with $C > 0$. Let $\mathcal{UO}$ be the set of user optimizable cost functions.*

Theorem 1. *If the cost function $\mathfrak{f}$ is not in $\mathcal{UO}$, then there exist connected evolving graphs where no solution exists for the ODOC space-time travel planning problem.*

Proof. First, it is clear that if $\mathfrak{f}(d) < 0$ for some $d \in \mathbb{N}^*$, then we can construct travels with arbitrarily small cost by repeatedly appending $((y,t),(y,t+d),(y,t))$ to any travel arriving at node y at time t (*i.e.*, by waiting for d rounds and going back in time d rounds), rendering the problem unsolvable.

Now, let $C \in \mathbb{N}^*$ and $\mathfrak{f}$ be a non-negative function that does not attain its minimum when restricted to $[C, \infty)$. This implies that there exists an increasing sequence $(w_i)_{i\in\mathbb{N}}$ of integers $w_i \geq C$, such that the sequence $(\mathfrak{f}(w_i))_{i\in\mathbb{N}}$ is decreasing and converges towards the lower bound $m_C = \inf_{t\geq C}(\mathfrak{f}(t))$ of $\mathfrak{f}_{|_{[C,\infty)}}$. Consider a graph with two nodes x_0 and x_1 that are connected by a temporal edge after time C and disconnected before. Since a travel from x_0 to x_1 arriving at time 0 must contain a backward travel to the past of amplitude at least C, its cost is at least equal to m_C. Since m_C is not attained, there is no travel with cost exactly m_C. Now, assume for the sake of contradiction that a cost-optimal travel T to x_1 arriving at time 0 has cost $m_C + \varepsilon$ with $\varepsilon > 0$. Then, we can construct a travel with a smaller cost. Let i_ε such that $\mathfrak{f}(w_{i_\varepsilon}) < m_C + \varepsilon$ (this index exists because the sequence $(\mathfrak{f}(w_i))_{i\in\mathbb{N}}$ converges to m_C).

Let $T' = ((x_0,0),(x_0,C),(x_1,C),(x_1,w_{i_\varepsilon}),(x_1,0))$. Then we have

$$cost(T') = \mathfrak{f}(w_{i_\varepsilon}) < m_C + \varepsilon = cost(T),$$

which contradicts the optimality of T. □

We now present the set of *user friendly* cost functions that we use in the sequel to ease proving optimization algorithms, as they allow *simple* solutions to the ODOC problem (Lemma 1). We prove in Theorem 2 that we do not lose generality since an algorithm solving the ODOC problem with user friendly cost functions can be transformed easily to work with any user optimizable ones.

Definition 11. *A cost function* $\mathfrak{f}$ *is* user friendly *if it is user optimizable, non-decreasing, and sub-additive*[4]*. Let* $\mathcal{UF}$ *be the set of user friendly cost functions.*

Lemma 1. *If the cost function* $\mathfrak{f}$ *is in* $\mathcal{UF}$ *and there exists a solution to the ODOC space-time travel planning problem in an evolving graph* G*, then there also exists a* simple *travel solution.*

Proof. Let T be a solution to the ODOC space-time travel planning problem. If there exists a node x_i and two time instants t_1 and t_2, such that $T = T_1 \oplus ((x_i, t_1)) \oplus T_2 \oplus ((x_i, t_2)) \oplus T_3$, then we construct T' as follows

$$T' = T_1 \oplus ((x_i, t_1), (x_i, t_2)) \oplus T_3$$

and we show that $cost(T') \leq cost(T)$. Indeed, it is enough to show (thanks to Remark 1) that

$$cost(((x_i, t_1), (x_i, t_2))) \leq cost(T_2).$$

By definition $cost(((x_i, t_1), (x_i, t_2))) = \mathfrak{f}(t_1 - t_2)$. If $t_1 < t_2$, then the cost is null by convention and the Lemma is proved. Otherwise $t_1 > t_2$. On the right hand side, we have:

$$cost(T_2) = \sum_{i=1}^{k} \mathfrak{f}(d_i)$$

where $d_1, d_2, \ldots, d_k$ is the sequence of differences between the times appearing in T_2. Since T_2 starts at time t_1 and ends at time t_2, then $\sum_{i=1}^{k} d_i = t_1 - t_2$. Since the function is sub-additive and increasing, we obtain:

$$\mathfrak{f}(t_1 - t_2) < \sum_{i=1}^{k} \mathfrak{f}(d_i)$$

By repeating the same procedure, we construct a time-travel with the same destination and same backward-cost as T but that does not contain two occurrences of the same node, except if they are consecutive. □

Theorem 2. *If an algorithm* A *solves the optimal cost space-time travel planning problem for any cost function in* $\mathcal{UF}$*, then there exists an algorithm* A' *solving the same problem with any* $\mathfrak{f}$ *in* $\mathcal{UO}$*.*

Proof. We consider an algorithm A as stated. Let $\mathfrak{f}$ be an arbitrary cost function in $\mathcal{UO}$, that is, $\mathfrak{f}$ is non-negative, and always attains its minimum.

From $\mathfrak{f}$, we now construct a cost function $\mathfrak{f}_{inc}$ as follows:

$$\mathfrak{f}_{inc}(t) = \min_{j \geq t} (\mathfrak{f}(j))$$

[4] sub-additive means that for all $a, b \in \mathbb{N}$, $\mathfrak{f}(a + b) \leq \mathfrak{f}(a) + \mathfrak{f}(b)$

By construction, $\mathfrak{f}_{inc}$ is non-decreasing. Moreover, since $\mathfrak{f}$ is in $\mathcal{UO}$, it always attains its minimum, and we have:

$$\forall d,\ \exists d_m \text{ such that } \mathfrak{f}_{inc}(d) = \mathfrak{f}(d_m). \tag{1}$$

Then, we construct $\widetilde{\mathfrak{f}}$ as follows:

$$\widetilde{\mathfrak{f}}(t) = \min_{a \in \alpha(t)} \left(\sum_{a_i \in a} \mathfrak{f}_{inc}(a_i) \right)$$

where $\alpha(t)$ is the set of all the non-negative sequences that sum to t. Now, $\widetilde{\mathfrak{f}}$ is sub-additive by construction, hence $\widetilde{\mathfrak{f}} \in \mathcal{UF}$. Since $\alpha(t)$ is finite, the minimum is attained.

Also, $\forall t \geq 1, \widetilde{\mathfrak{f}}(t) \leq \mathfrak{f}(t)$, so that for any travel, its backward cost with respect to $\mathfrak{f}$ is at least equal to its backward cost with respect to $\widetilde{\mathfrak{f}}$.

Let G be a dynamic graph. Our goal is to construct an algorithm A' finding a cost-optimal (with respect to $\mathfrak{f}$) space-time travel in G. The algorithm A' works as follows. Let $\widetilde{T}$ be an optimal solution found by algorithm A on G assuming function $\widetilde{\mathfrak{f}}$ is used. A' now constructs, from $\widetilde{T}$, a time-travel T that is a cost-optimal (with respect to $\mathfrak{f}$) on G.

The travel T is constructed from $\widetilde{T}$ by replacing any sub-space-time travel $((x_i, t_i), (x_i, t_i - t))$, with $t \geq 0$, by the following sub space-time travel: $((x_i, t_i - a_1), (x_i, t_i - a_1 - a_2), \ldots, (x_i, t_i - \sum_{j=1}^{k} a_j))$ satisfying:

$$a \in \alpha(t) \quad \wedge \quad \widetilde{\mathfrak{f}}(t) = \sum_{j=1}^{\text{length of } a} \mathfrak{f}_{inc}(a_j)$$

Then, each $((u, t), (u, t - d))$, with $d \geq 0$, is replaced by $((u, t), (u, t - d + d_m), (u, t - d))$ such that:

$$d_m \geq d \quad \wedge \quad \mathfrak{f}_{inc}(d_m) = \mathfrak{f}(d)$$

We know that d_m exists thanks to Equation 1. The space-time travel T uses the same temporal edges as $\widetilde{T}$, so it is well defined. Moreover, by construction $\mathfrak{f}(T) = \widetilde{\mathfrak{f}}(\widetilde{T})$, and T is optimal with respect to $\mathfrak{f}$ because the backward-cost of a travel with respect to $\mathfrak{f}$ is at least equal to its backward-cost with respect to $\widetilde{\mathfrak{f}}$, as observed earlier. Hence, if a better solution exists for $\mathfrak{f}$, it is also a solution with the same, or smaller, cost with $\widetilde{\mathfrak{f}}$, contradicting the optimality of $\widetilde{T}$. The above procedure defines an algorithm, based on A, that solves the ODOC problem with function $\mathfrak{f}$. □

4 Offline $\mathcal{C}$-cost-constrained ODOC Algorithm

In this section, we present Algorithm 1 that solves the $\mathcal{C}$-cost-constrained ODOC problem in time polynomial in the number of edges. More precisely, since the

Algorithm 1: Offline $\mathcal{C}$-cost-constrained ODOC Algorithm (input: $G, \mathfrak{f}, \mathcal{C}, src, dst$)

```
   /* nodeCost[u,t] stores the current best cost of travels from node
      src to node u arriving at time t.  minCost[u] stores a pair (c,t)
      where c is the current known minimum cost of a travel towards u,
      and t the smallest time where such travel arrives. pred[u,t]
      stores the suffix of an optimal travel to u arriving at t.     */
 1 ∀u ∈ V, ∀t,   nodeCost[u,t] = ∞   minCost[u] = (∞,∞);
 2 nodeCost[src,0] ← 0;        done ← ∅;
 3 while ∃(u,t) ∉ done such that nodeCost[u,t] < ∞ do
 4     (u,t) ← argmin_{(u,t)∉done}(nodeCost[u,t]) ;
 5     done ← done ∪ {(u,t)};
 6     c ← nodeCost[u,t];
 7     for each neighbor v of u do
 8         let t_future the smallest time after (or equal to) t where edge
             ((u,v), t_future) exists;
 9         let (c_min, t_min) = minCost[v];
10         if nodeCost[v,t_future] > c and (c < c_min or t_future < t_min) then
11             nodeCost[v,t_future] ← c;
12             pred[v,t_future] ← ((u,t),(u,t_future),(v,t_future));
13             if (c,t_future) <_lexico minCost[v] then minCost[v] ← (c,t_future) ;
14         for each t_past such that (u,v) ∈ E_{t_past} do
15             let c_past = c + f(t − t_past);
16             if c_past ≤ C and nodeCost[v,t_past] > c_past then
17                 nodeCost[v,t_past] ← c_past;
18                 pred[v,t_past] ← ((u,t),(u,t_past),(v,t_past));
19 let t_min be the minimum time instant such that ∃t,
     nodeCost[dst,t] + f(t − t_min) ≤ C;
20 if t_min exists then return ExtractTimeTravel(dst, t_min, nodeCost, pred);
21 else return ⊥ ;
```

number of edges can be infinite, we only consider edges occurring before a certain travel (see the end of the section for a more precise description of the complexity). Algorithm 1 is different from existing shortest path algorithms because we need to efficiently take into account the cost and the delay of travels. It is well-known that constrained shortest path algorithms are exponential when considering two additive metrics [15] but surprisingly, our algorithm is polynomial by using the specificity of the time travel. Our algorithm works as follows. At each iteration, we extract the minimum cost to reach a particular node at a particular time and we extend travels from there by updating the best-known cost of the next node. We reach the next nodes either by using the next temporal edge that exists in the future (we prove that considering only the next future edge is enough) or using each of the past temporal edge.

We first prove that our algorithm terminates, even if the graph is infinite and if there is no solution.

Lemma 2. *Algorithm 1 always terminates.*

Proof. Assume for the sake of contradiction that it does not terminates. First, we observe that, for any $u \in V$, $\texttt{minCost}[u]$ is non-increasing (using the lexicographical order), so it must reach a minimum value $(c_{u,\min}, t_{u,\min})$, which represent, for a node u, the minimum cost a travel towards u can have and the minimum time such a travel can arrive. Moreover, the cost associated with a pair (u,t) extracted in Line 4 is non-decreasing (because we always extract a pair with minimum cost), so either this cost reach a maximum or tends to infinity. In the former case, let $c_{\max}$ be that maximum *i.e.*, after some time, every time a pair (u,t) is extracted, $\texttt{nodeCost}[u,t] = c_{\max}$. Since a pair is never extracted twice, pairs are extracted with arbitrarily large value t. Some, at some point in the execution, for every pair (u,t) extracted, we have $t > t_{u,\min}$. Moreover, $c_{\max} \geq c_{u,\min}$. So, every time a pair is extracted, condition Line 10 is false. Hence, $c_{\max}$ is not added into `nodeCost` anymore, which contradicts the fact that $c_{\max}$ is associated with each extracted pair after some time. So the latter case occurs *i.e.*, the cost associated with extracted pairs tends to infinity. After some time, this cost is greater than any $c_{u,\min}$. Again, since a pair is never extracted twice, pairs are extracted with arbitrarily large value t. Some, at some point in the execution, for every pair (u,t) extracted, we have $t > t_{u,\min}$, and the condition Line 10 is always false. Hence, from there, every time a value is added into `nodeCost`, it is according to Line 17, so the associated time smaller than the time extracted, which contradicts the fact that arbitrarily large value t are added to `nodeCost`. □

We now prove the correctness of our algorithm, starting with the main property we then use to construct a solution. Let $\delta_{\mathcal{C}}$ be the function that returns, for each pair (u,t) where u is a node and t a time, the best backward-cost smaller or equal to $\mathcal{C}$, from *src* to u, for travels arriving at time t.

Lemma 3. *When a pair (u,t) is extracted from `nodeCost` at line 4, then*

$$\delta_{\mathcal{C}}(u,t) = \texttt{nodeCost}[u,t]$$

Proof. Assume for the sake of contradiction that this is not true, and let (u,t) be the first tuple extracted such that the property is false. Let $c_{u,t} = \texttt{nodeCost}[u,t]$. Let T be a $\mathcal{C}$-cost-constrained backward-cost-optimal travel to u arriving at time t (hence $cost(T) < c_{u,t}$ by assumption).

Let T' be the longest prefix of T, to (x,t') (*i.e.*, such that $T = T' \oplus (x,t') \oplus T''$, for some T''), such that (x,t') was extracted from `nodeCost` and satisfies $\delta_{\mathcal{C}}(x,t') = \texttt{nodeCost}[x,t']$. Now, T' is well defined because the first element in T is $(src,0)$ and, by Line 2, $(src,0)$ is the first extracted pair, and satisfies $\texttt{nodeCost}[src,0] = 0 = \delta_{\mathcal{C}}(src,0)$. Hence, prefix $((src,0))$ satisfies the property, so the longest of such prefixes exists. Observe that T', resp. T'', ends, resp. starts, with (x,t'), by the definition of travel concatenation.

When (x, t') is extracted from `nodeCost`, it is extended to the next future edge (Lines 8 to 11), and all past edges (Lines 14 to 17). T'' starts either (a) with $((x, t'), (x, t_a), (y, t_a))$, with $t_a < t'$, (b) with $((x, t'), (x, t_a), (y, t_a))$ with $t_a > t'$, or (c) with $((x, t'), (y, t'))$, where $y \in N(x)$.

In case (a)**,** this means that the temporal edge $((x, y), t_a)$ exists, hence, by Line 17, we know that $\texttt{nodeCost}[y, t_a] \leq \texttt{nodeCost}[x, t'] + \mathfrak{f}(t' - t_a)$. However, since T' is a sub-travel, $cost(T') = \delta_{\mathcal{C}}(x, t') = \texttt{nodeCost}[x, t']$, hence

$$\texttt{nodeCost}[y, t_a] \leq cost(T' \oplus ((x, t'), (x, t_a), (y, t_a))) = \delta_{\mathcal{C}}(y, t_a),$$

and (y, t_a) must have been extracted before (u, t), otherwise

$$\delta_{\mathcal{C}}(u, t) < \texttt{nodeCost}[y, t] \leq \texttt{nodeCost}[y, t_a] = \delta_{\mathcal{C}}(y, t_a)$$

which is a contradiction (a sub-travel of a cost-optimal travel cannot have a greater cost, see Property 1). So, $T' \oplus ((x, t'), (x, t_a), (y, t_a))$ is a longer prefix of T with the same property as T', which contradicts the definition of T'.

In case (b)**,** this means that the temporal edge $((x, y), t_a)$ exists, hence, by Line 11, we know that $\texttt{nodeCost}[y, t_a] \leq \texttt{nodeCost}[x, t']$. Again, we have $cost(T') = \delta_{\mathcal{C}}(x, t') = \texttt{nodeCost}[x, t']$, hence

$$\texttt{nodeCost}[y, t_a] \leq cost(T' \oplus ((x, t'), (x, t_a), (y, t_a))) = \delta_{\mathcal{C}}(y, t_a),$$

which contradicts the definition of T'.

In case (c)**,** this means that the edge $((x, y), t')$ exists, which implies, using a similar argument, a contradiction. □

The previous lemma says that `nodeCost` contains correct information about the cost to reach a node, but actually, it does not contain all the information. Indeed, a node u can be reachable by a travel at a given time t and still $\texttt{nodeCost}[u, t] = \infty$. This fact helps our algorithm to be efficient, as it does not compute all the optimal costs for each possible time (in this case, the complexity would depend on the duration of the graph, which could be much higher than the number of edges). Fortunately, we now prove that we can still find all existing travel using `nodeCost`.

Lemma 4. *For all $u \in V$, $t \in \mathbb{N}$, there exists a $\mathcal{C}$-cost-constrained travel T from src to u arriving at time t, if and only if there exists $t' \in \mathbb{N}$ such that $\texttt{nodeCost}[u, t'] + \mathfrak{f}(t' - t) \leq \mathcal{C}$.*

Theorem 3. *If the cost function $\mathfrak{f}$ is in $\mathcal{UF}$, Algorithm 1 outputs a travel T if and only if T is a solution of the $\mathcal{C}$-cost-constrained ODOC problem.*

Let us now analyze the complexity of Algorithm 1. We assume that retrieving the next or previous edge after or before a given time takes $O(1)$ time. For example, the graph can be stored as a dictionary that maps each node to an array that maps each time to the current, previous, and next temporal edges.

This array can be made sparser with low complexity overhead to save space if few edges occur per time-instant.

Since each temporal edge is extracted from `nodeCost` at most once and the inner *for* loop iterates over a subset of edges, the time complexity is polynomial in the number of temporal edges. We must also consider the time to extract the minimum from `nodeCost`, which is also polynomial. If there are an infinite number of temporal edges[5], Lemma 2 shows that our algorithm always terminates, even if no solution exists. Therefore, its complexity is polynomial in the size of the finite subset of temporal edges extracted from `nodeCost`.

Let $\mathcal{E}$ be the set of temporal edges $((u,v),t)$ such that (u,t) or (v,t) is extracted in Line 4 of our algorithm during its execution.

Theorem 4. *If the cost function* $\mathfrak{f}$ *is in* $\mathcal{UF}$*, then Algorithm 1 terminates in* $O(|\mathcal{E}|^2)$*.*

5 Offline $\mathcal{H}$-history-constrained ODOC Algorithm

Section 4 made the assumption that a given agent was able to go back to *any* previous snapshot of the network. However, this hypothesis might not hold as the difficulty to go back in time may depend on how far in the future we already reach. Hence, we consider in this section that $\mathcal{H}$ denotes the maximum number of time instants one agent can travel back to. In more detail, once an agent reaches time instant t, it cannot go back to $t' < t - \mathcal{H}$, even after multiple jumps.

In this section, it is important to notice that the capability of BTT devices does not depend on the time when the agent uses it but rather on the largest time reached by the agent.

We present Algorithm 2 that solve the $\mathcal{H}$-history-constrained ODOC problem. The algorithm uses dynamic programming to store intermediary results. At each iteration, we update the optimal cost based on the best cost of previous nodes. For each node x_i and time t we need to store the best cost depending on the maximum time reached by the agent.

Theorem 5. *If the cost function* $\mathfrak{f}$ *is in* $\mathcal{UF}$*, then Algorithm 2 solves the* $\mathcal{H}$*-history-constrained ODOC problem and has* $O(n^2\mathcal{H}(t_{\min}+\mathcal{H}))$ *complexity, with* $t_{\min}$ *the delay of a solution.*

6 Conclusion

We presented the first solutions to the optimal delay optimal cost space-time constrained travel planning problem in dynamic networks, and demonstrated that the problem can be solved in polynomial time, even in the case when backward time jumps can be made up to a constant, for any sensible pricing policy. It would be interesting to investigate the online version of the problem, when the future of the evolving graph is unknown to the algorithm.

[5] An evolving graph with an infinite number of edges can exist in practice even with bounded memory, e.g., when the graph is periodic.

Algorithm 2: Offline $\mathcal{H}$-history-constrained ODOC Algorithm

```
/* c[i, t−h, t] stores the cost of a cost optimal travel to node x_i,
   arriving before or at time t−h, that is H-history-constrained,
   and never reaches a time instant greater than t.
   pred[u, t−h, t] stores the suffix of an optimal travel to u
   arriving at t−h that never reaches a time greater than t.      */
c[∗] ← ∞;   c[src, ∗] ← 0   pred[∗] ← ⊥;
t_max ← upper bound on the time reached by a cost-optimal travel to dst;
for t = 0, 1, 2, ..., t_max do
    /* for simplicity, we assume c[u, t−h, t] = ∞ if t−h < 0        */
    for u ∈ V do
        c[u, t−h, t] ← min(c[u, t−h, t−1], c[u, t−h−1, t−1]);
    repeat |V| times
        for u ∈ V do
            for h = H, H−1, ..., 0 do
                m ← min_{t'∈[t−H,t], (u,v)∈E_{t'}} (c[v, t', t] + f(t' − (t−h)));
                if c[u, t−h, t] < m then
                    c[u, t−h, t] ← m;
                    pred[u, t−h, t] ← (v, t') (with the corresponding min
                      arguments);
    if the minimum time instant t_min such that c[dst, t_min, t_min + H] < ∞
      exists then
        return ExtractHistoryConstrainedTravel(dst, t_min, t_min + H, c);
return ⊥;
```

References

1. Casteigts, A., Flocchini, P., Mans, B., Santoro, N.: Shortest, fastest, and foremost broadcast in dynamic networks. Int. J. Found. Comput. Sci. **26**(4), 499–522 (2015), https://doi.org/10.1142/S0129054115500288
2. Casteigts, A., Flocchini, P., Quattrociocchi, W., Santoro, N.: Time-varying graphs and dynamic networks. Int. J. Parallel Emergent Distributed Syst. **27**(5), 387–408 (2012), https://doi.org/10.1080/17445760.2012.668546
3. Casteigts, A., Himmel, A., Molter, H., Zschoche, P.: Finding temporal paths under waiting time constraints. Algorithmica **83**(9), 2754–2802 (2021), https://doi.org/10.1007/s00453-021-00831-w
4. Casteigts, A., Peters, J.G., Schoeters, J.: Temporal cliques admit sparse spanners. J. Comput. Syst. Sci. **121**, 1–17 (2021), https://doi.org/10.1016/j.jcss.2021.04.004
5. Chen, S., Nahrstedt, K.: An overview of qos routing for the next generation high-speed networks: Problems and solutions. Network, IEEE **12**, 64 – 79 (12 1998). https://doi.org/10.1109/65.752646
6. Ferreira, A.: On models and algorithms for dynamic communication networks: The case for evolving graphs. In: Quatrièmes Rencontres Francophones sur les Aspects

Algorithmiques des Télécommunications (ALGOTEL 2002). pp. 155–161. INRIA Press, Mèze, France (May 2002)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., USA (1990)
8. Garroppo, R.G., Giordano, S., Tavanti, L.: A survey on multi-constrained optimal path computation: Exact and approximate algorithms. Computer Networks **54**(17), 3081–3107 (Dec 2010). https://doi.org/10.1016/j.comnet.2010.05.017
9. Guck, J.W., Van Bemten, A., Reisslein, M., Kellerer, W.: Unicast qos routing algorithms for sdn: A comprehensive survey and performance evaluation. IEEE Communications Surveys & Tutorials **20**(1), 388–415 (2018). https://doi.org/10.1109/COMST.2017.2749760
10. Luna, G.A.D., Flocchini, P., Prencipe, G., Santoro, N.: Black hole search in dynamic rings. In: 41st IEEE International Conference on Distributed Computing Systems, ICDCS 2021, Washington DC, USA, July 7-10, 2021. pp. 987–997. IEEE (2021), https://doi.org/10.1109/ICDCS51616.2021.00098
11. Pal, G.: The time machine (1960)
12. Russo, A., Russo, J.: Avengers: Endgame (2019)
13. Thulasiraman, K., Arumugam, S., Brandstädt, A., Nishizeki, T.: Handbook of graph theory, combinatorial optimization, and algorithms (2016)
14. Villacis-Llobet, J., Bui-Xuan, B., Potop-Butucaru, M.: Foremost non-stop journey arrival in linear time. In: Parter, M. (ed.) Structural Information and Communication Complexity - 29th International Colloquium, SIROCCO 2022, Paderborn, Germany, June 27-29, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13298, pp. 283–301. Springer (2022), https://doi.org/10.1007/978-3-031-09993-9_16
15. Wang, Z., Crowcroft, J.: Quality-of-service routing for supporting multimedia applications. IEEE Journal on Selected Areas in Communications **14**(7), 1228–1234 (Sep 1996). https://doi.org/10.1109/49.536364
16. Zemeckis, R.: Back to the future (1985)